下载APP

11｜深度学习（上）：用CNN带你认识深度学习

2021-09-22 黄佳

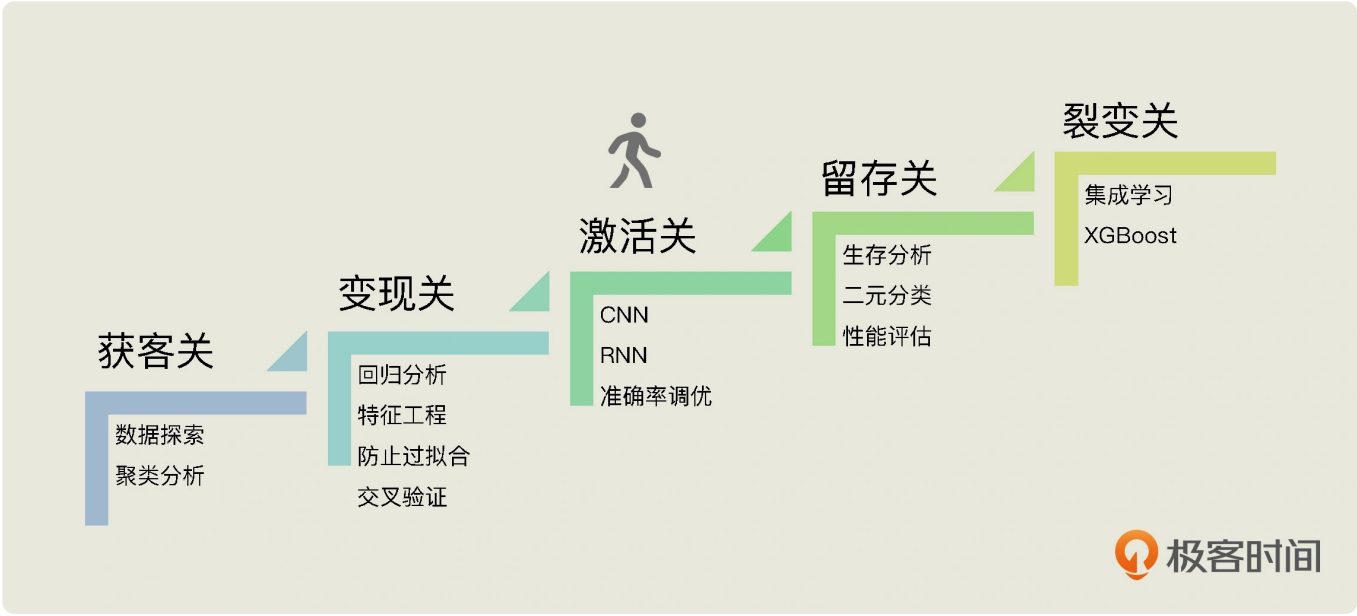
《零基础实战机器学习》 课程介绍 >

讲述：黄佳

时长 27:21 大小 25.06M

你好，我是黄佳。

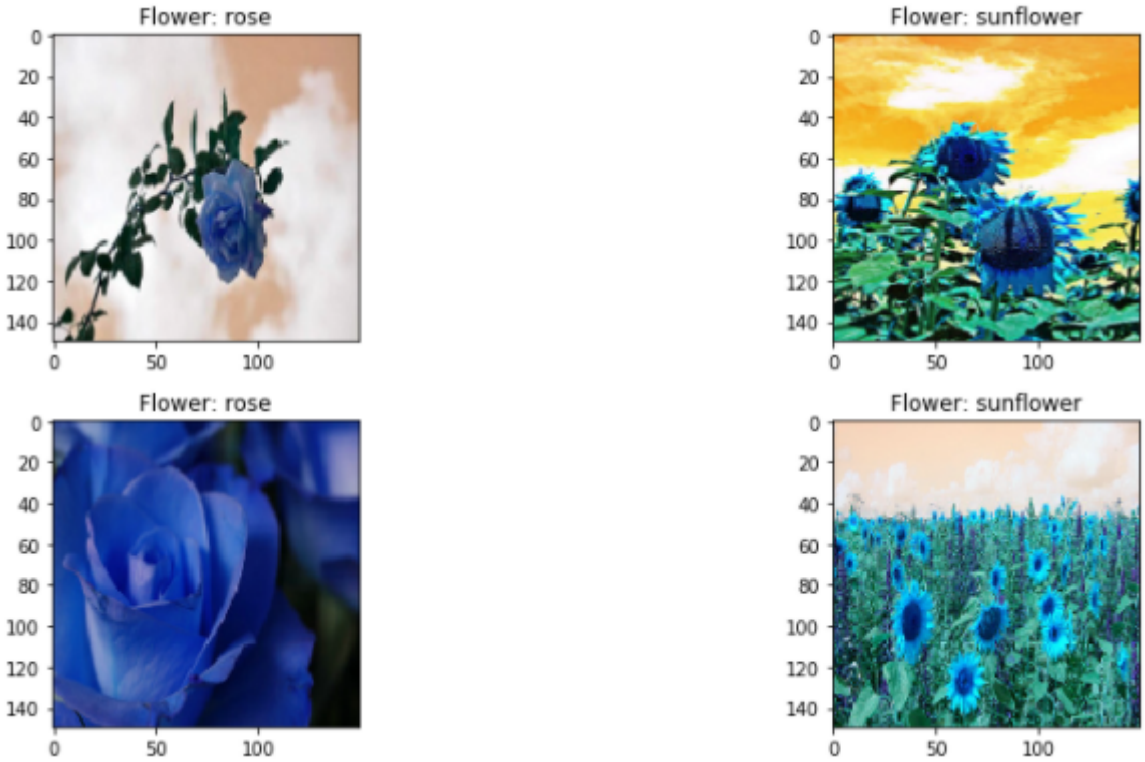
随着课程的不断深入，我们已经成功闯过两个关卡，获客关和变现关，学到了多种机器学习算法和模型的优化方法。今天这一讲，我们正式进入“激活关”。在这一关中，我们的主要任务是根据易速鲜花 App 的历史记录，借助深度学习神经网络，来预测它的日激活数趋势。通过“激活关”，相信你会对深度学习神经网络的原理和用法有一个比较深入的理解。



不过，在正式进入这个项目之前呢，我们需要先打打基础，解决两个问题：1. 深度学习的原理是什么？2. 怎么搭建起一个深层神经网络 CNN？这两个问题，我们将通过一个完整的小项目来搞定。

问题定义

易速鲜花的供应商每天都会发来大量的鲜花图片，不过，这些图片都没有按鲜花的类别进行分门别类，要是由人工来做的话，比较麻烦，成本也比较高。现在，我们需要根据易速鲜花大量已归类的鲜花图片，来建立一个能识别鲜花的模型，给未归类的图片自动贴标签。



这是一个典型的分类问题，同时也是一个计算机视觉领域的图片识别问题。那么下面我们先来看看在哪里可以找到这些花的图片。

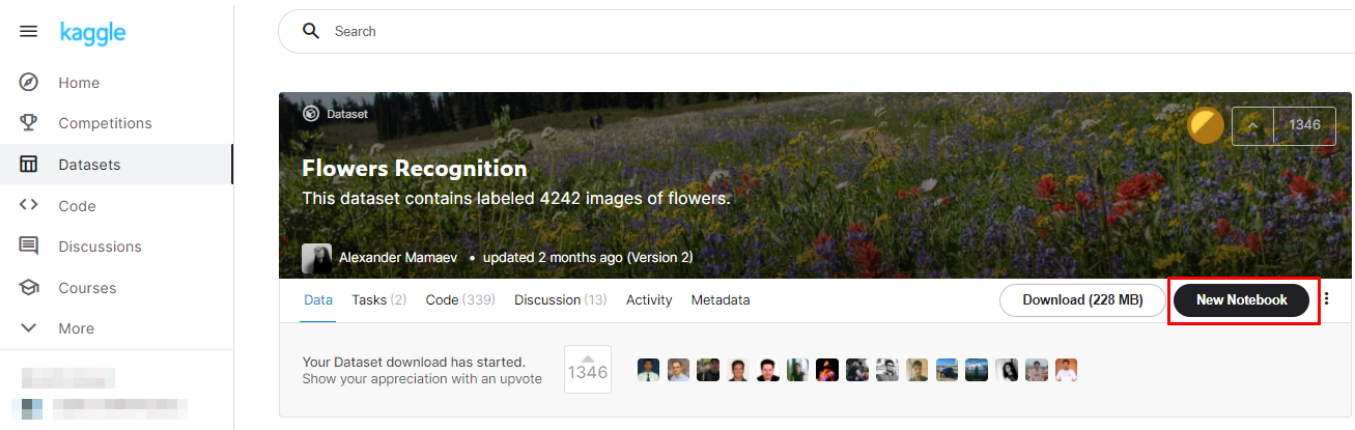
数据收集和预处理

不知道你是否还记得，在第 2 讲中我曾经说过，在深度学习的实战部分，我会带你去 Kaggle 网站用一用 GPU、TPU 等加速器。Kaggle 网站是一个在线 Jupyter Notebook 平台，同时也是数据科学爱好者最好的学校交流场所，里面有数据集、源代码、课程资料等。我们这个项目会在 Kaggle 网站上完成。

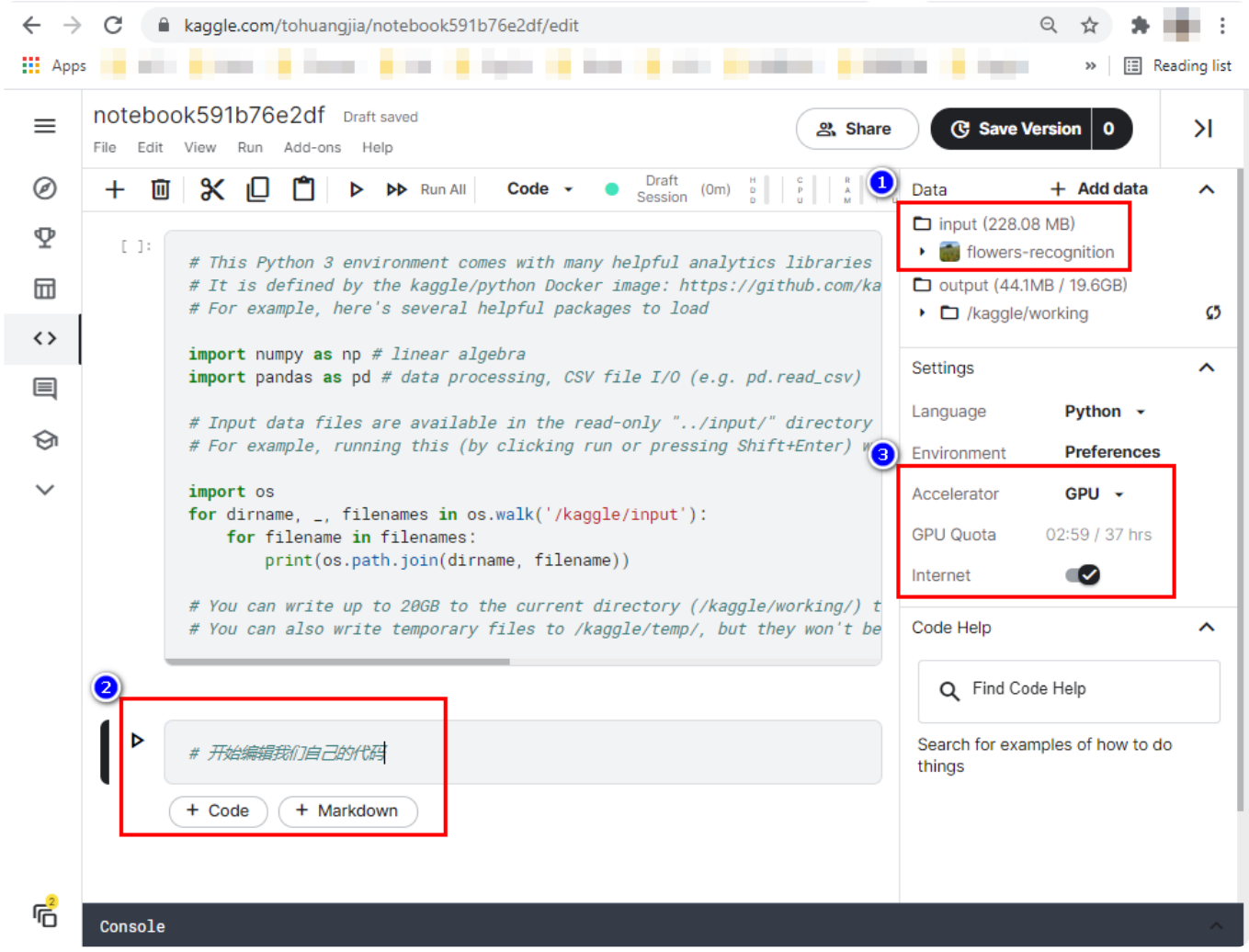
1. 数据的导入及可视化

在 Kaggle 网站上，有一个公开的数据集，里面收纳了各类花朵图片，我们可以将它作为这个项目的数据集。由于这个花朵图片的数据集较大，我们不必把它下载下来，可以直接在网站中创建 Notebook，来完成对花朵图片进行归类的工作。

首先，请你点开这个花朵数据集，单击 New Notebook，建立你在 Kaggle 网站上的第一个深度学习 Notebook。如果你没有用过这个网站，那在这个过程中，肯定需要你新注册用户，你跟着网站的说明走就可以了。当然了，你也可以访问我公开发表的 Notebook，并 Copy & Edit 我的代码。



新建 Notebook 之后，你就会来到像这样的页面，这个页面的基本操作和我们在本机安装的 Jupyter Notebook 完全一致，它其实就是网页版的 Jupyter Notebook：



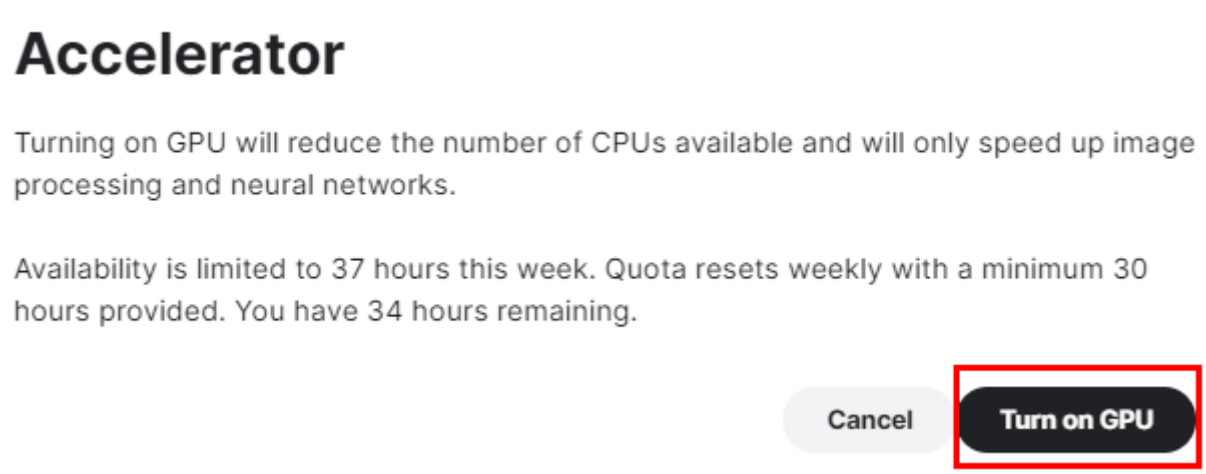
这里我要提醒你注意 3 个地方，就是我在图中用红框标出的：

右上角红框里的 Input，代表我们当前正在使用的数据集，基本上每一个 Notebook 都需要基于相关数据集创建；

左下角点击“+ Code”，可以生成一段新的代码单元；

右侧第二个红框是 GPU 加速选项，在跑深度学习项目时，你可以考虑打开它，如果你新注册了 Kaggle，就会有 30 多个小时的 GPU 使用时间，跑完项目后你要记得关掉。还有就是，在安装新的工具包的时候，要打开 Internet 选项，不然，无法 pip install 新的包。

在打开 GPU 选项的时候，会提示你每周可以使用 GPU 的小时数，选择“Turn on GPU”就可以了。


Accelerator
Turning on GPU will reduce the number of CPUs available and will only speed up image processing and neural networks.
Availability is limited to 37 hours this week. Quota resets weekly with a minimum 30 hours provided. You have 34 hours remaining.
Cancel
Turn on GPU

下面，我们指定 4 个花朵目录，并通过 Open CV（开源计算机视觉库）工具箱，读入图片的数据。OpenCV 是一个跨平台的开源计算机视觉方面的 API 库，这里我们应用其中的 imread 和 resize 功能读入并裁剪图片到 150*150 像素：

复制代码

```
import numpy as np # 导入Numpy
import pandas as pd # 导入Pandas
import os # 导入OS
import cv2 # 导入Open CV工具箱


print(os.listdir('../input/flowers-recognition/flowers')) #打印目录结构
daisy_dir='../input/flowers-recognition/flowers/daisy' #雏菊目录
rose_dir='../input/flowers-recognition/flowers/rose' #玫瑰目录
sunflower_dir='../input/flowers-recognition/flowers/sunflower' #向日葵目录
tulip_dir='../input/flowers-recognition/flowers/tulip' #郁金香目录


X = [] #初始化
y_label = [] #初始化
imgsize = 150 #图片大小
# 定义一个函数读入花的图片
def training_data(label,data_dir):
    print ("正在读入：", data_dir)
    for img in os.listdir(data_dir): #目录
        path = os.path.join(data_dir,img) #目录+文件名
        img = cv2.imread(path,cv2.IMREAD_COLOR) #读入图片
        img = cv2.resize(img,(imgsize,imgsize)) #设定图片像素维度
        X.append(np.array(img)) #X特征集
        y_label.append(str(label)) #y标签，即花的类别
# 读入目录中的图片
training_data('daisy',daisy_dir) #读入雏菊
training_data('rose',rose_dir) #读入玫瑰
training_data('sunflower',sunflower_dir) #读入向日葵
training_data('tulip',tulip_dir) #读入郁金香
```

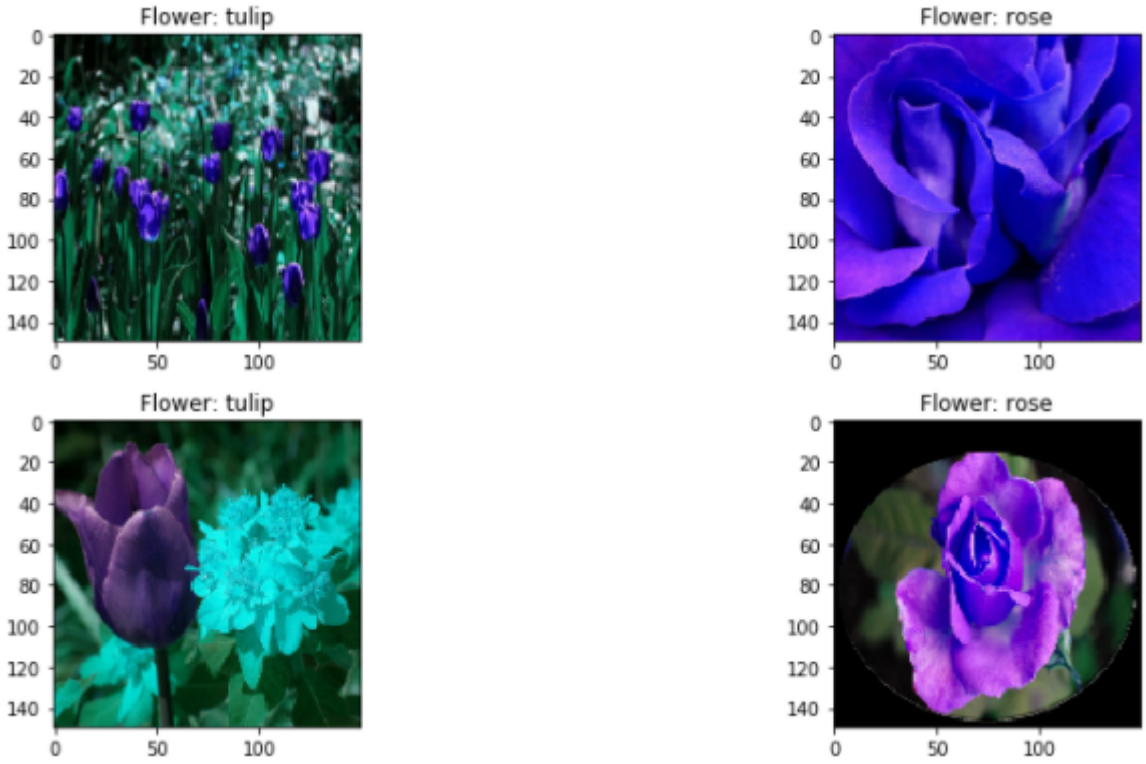
输出如下：

```
正在读入： ../input/flowers/daisy
正在读入： ../input/flowers/sunflower
正在读入： ../input/flowers/tulip
正在读入： ../input/flowers/rose
```

图片数据导入程序之后，我们随机用 imshow 功能显示几张花朵的图片，来确认一下我们已经成功读入了图片。

```
import matplotlib.pyplot as plt # 导入matplotlib
import random as rdm # 导入随机数工具
# 随机显示几张漂亮的花朵图片吧
fig,ax=plt.subplots(5,2) #画布
fig.set_size_inches(15,15) #大小
for i in range(5):
    for j in range (2):
        r=rdm.randint(0,len(X)) #随机选择图片
        ax[i,j].imshow(X[r]) #显示图片
        ax[i,j].set_title('Flower: '+y_label[r]) #花的类别
plt.tight_layout() #绘图
```

输出如下：



好，到这里呢，按照常规的机器学习项目步骤，紧接着下一个数据预处理环节就是数据清洗了，不过我们这个数据集的质量不错，没有需要清洗的内容，所以我们直接来构建特征集 X 和标签集 y。

2. 构建特征集和标签集

在下面的代码中，我们用 LabelEncoder 给标签 y 编码，并且把特征集 X 转换为了张量数组：

复制代码

```
from sklearn.preprocessing import LabelEncoder # 导入标签编码工具
from keras.utils.np_utils import to_categorical # 导入One-hot编码工具
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y_label) # 标签编码
y = to_categorical(y,10) # 将标签转换为One-hot编码
X = np.array(X) # 将X从列表转换为张量数组
```

请你注意，这时候特征集 X 的格式不再是 DataFrame 表结构，而是 NumPy 数组，在机器学习里，我们把它称为“张量”，它的形状输出如下：

复制代码

```
array([[[[214, 237, 233],
         [224, 234, 235],
```

```
3          [229, 232, 237],
4          ...,
5          [ 67,  93, 124],
6          [ 63,  91, 121],
7          [ 61,  93, 115]]]], dtype=uint8)
```

如果用 NumPy 里面 shape（就是张量的形状）的属性，我们就会看到当前特征集 X 的格式是 4 阶的张量。

复制代码

```
1 X.shape
```

输出如下：

复制代码

```
1 (3265, 150, 150, 3)
```

这个输出结果表示，在当前这个数据集中，一共有 3265 张 150 像素 *150 像素的图片，且所有图片的颜色通道数为 RGB 3。

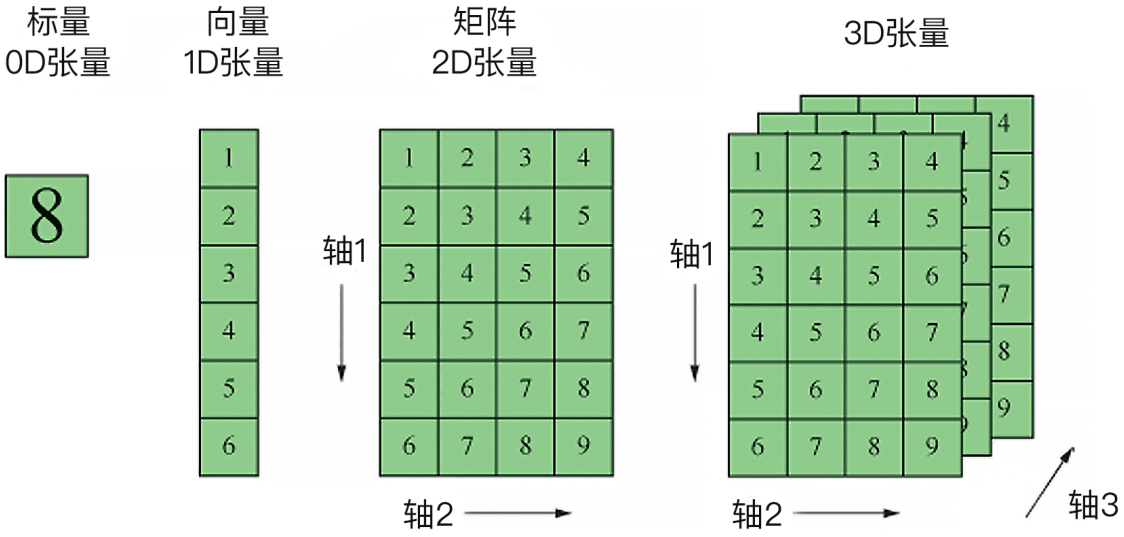
相信你已经体会到了，我们这里说的“4 阶张量”代表了图片数据集中的 4 个维度：行（图片数量）、宽（像素宽）、高（像素高）和颜色通道数，缺少其中任何一个，都无法精准描述这个图片数据集。这就是为什么我们当前这个特征集 X 不再是我们之前常见的二维 DataFrame 表结构，而是 NumPy 数组，也就是深度学习中的“张量”。

如果是视频格式的数据集，则需要 5 阶张量才放得下，其形状为（样本，帧，高度，宽度，颜色深度）。此外，文本数据集通常是 3 阶张量，形状为（样本，序号，字编码）

其实，从严格意义上讲，前几节课中那些的二维 DataFrame 数据表格，也就是一行一列，也都属于 2 阶张量。

那为什么我们这里强调“阶”，而不再叫“维度”呢？这是因为在机器学习中，我们又把每一个阶上的特征的个数称为“特征的维度”。所以，为了避免混淆，当我们提到张量时，就称其为几**阶**张量；而提到特征时，就称特征的**维度**是多少。

对于这个花朵图片的数据集来说，在图片“宽度”这阶特征上面，就有 150 个维度，对应着 150 个值；而“高度”这阶特征呢，也是 150 个维度；宽、高组合，就形成了 150*150，共这么多个像素点。这里面有点绕，你可以停下来体会一下，也可以看看下面的图。



各种张量

现在你应该明白了，为什么像图片、视频、文本这样的数据集，特征的维度和整体特征空间，体量都如此巨大。就这么一张小小的 150 * 150 像素的 RGB 图片，特征就有可能达到 150 * 150 * 3 的天文数字。所以，除了深层神经网络之外，传统机器学习方法根本解决不了。

而此时，y 的格式也转换成了 One-hot 编码的张量。至于什么是 One-hot 编码，我们在第 8 讲中介绍过，我在这里就不重复了。

复制代码

```
array([[1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
       ...,
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.]], dtype=float32)
```

其中，[1., 0., 0., 0.]代表 Daisy（雏菊），[0., 1., 0., 0.]就代表 Rose（玫瑰），[0., 0., 1., 0.]就代表 Sunflower（向日葵），[0., 0., 0., 1.]就代表 Tulip（郁金香）。

至此，我们的特征集 X 和标签集 y 就构建完毕。

3. 特征工程和数据集拆分

我们在第 1 讲中说过，深层神经网络的厉害之处在于，它能对非结构的数据集进行自动的复杂特征提取，完全不需要人工干预。因此，我们并不用做什么复杂的特征工程处理。

不过，由于神经网络特别喜欢小范围的数值，这里我们只需要做个归一化处理，把 0-255 的 RPG 像素值压缩到 0-1 之间最好。这个步骤非常重要，不然神经网络会跑不起来：

复制代码

```
X = X/255 # 将X张量归一化
```

接下来，就是数据集的拆分了，拆分的代码相信你已经非常熟悉了：

复制代码

```
from sklearn.model_selection import train_test_split # 导入拆分工具
X_train, X_test, y_train, y_test = train_test_split(X, y, #拆分数据集
                                test_size=0.2,random_state=1)
```

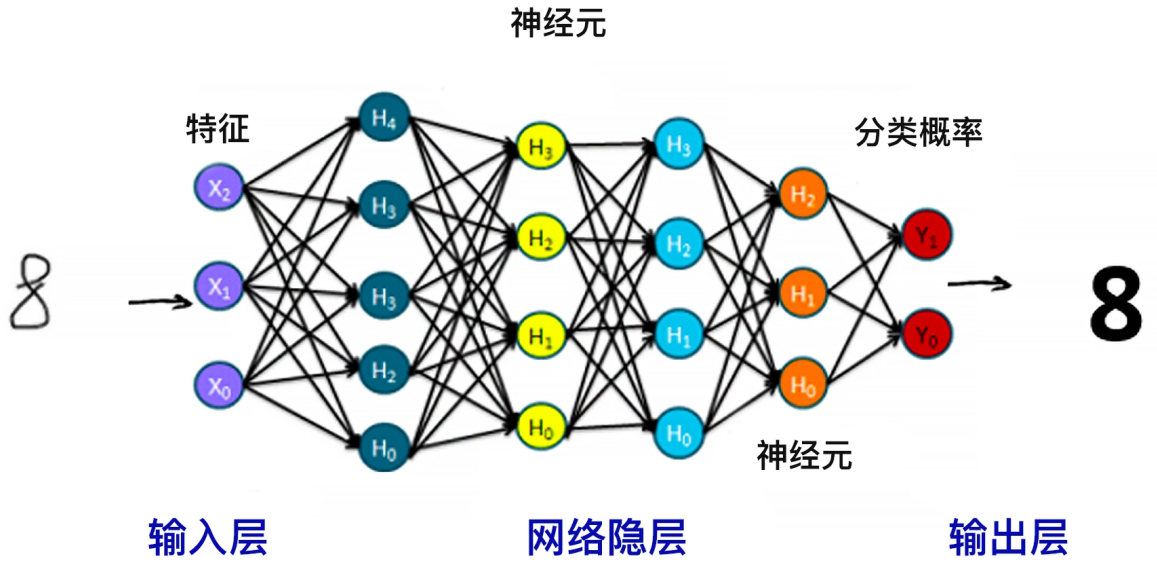
那到这里呢，我们就完成了数据集的所有准备工作，下面进入关键的建立模型环节。

选择算法建立模型

就我们的鲜花分类来说，选择算法的过程在所有问题中是最简单的。因为对于图像分类识别问题来说，深度学习中的卷积神经网络 CNN 是不二之选。

1. 选择算法

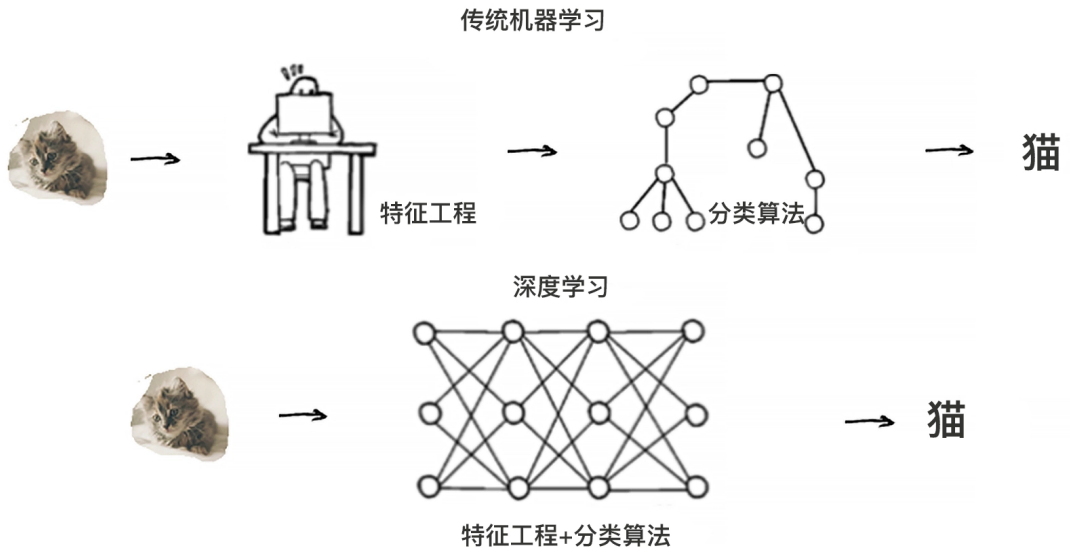
我们曾经说过，深层神经网络是由大量的人工神经元相互联结而成，这些神经元都具有可以调整的参数，而训练机器、建立模型的过程，也就是确定网络参数的过程。一旦参数确定，模型也就确定下来了。关于深度神经网络的结构，你可以看看下面的图。



那么你还记不记得，深度学习和传统机器学习算法相比，它的优势在哪里呢？

没错，深度学习特别擅长处理非结构化的数据。我们前面的每一个实战所用的数据集，都是具有良好的结构，每一个特征是什么，我们都能说得很清楚。但是，还有另外一大类数据是没有良好结构的，比如说图形图像数据、文本数据等，这些数据集的特征长什么样呢？不太容易说清楚。

所以，传统的模型需要先做各种各样的特征工程，让数据变得“计算机友好”，再输入模型进行学习。而深度学习模型则可以自动进行特征提取，因此就省略掉了手工做特征工程的环节。你可以看一看图中所示的这个图片识别问题的机器学习流程。



那么既然鲜花分类的算法易于确定，下面我们看看怎么选择一个好用的深度学习包（也叫框架）来创建神经网络模型。

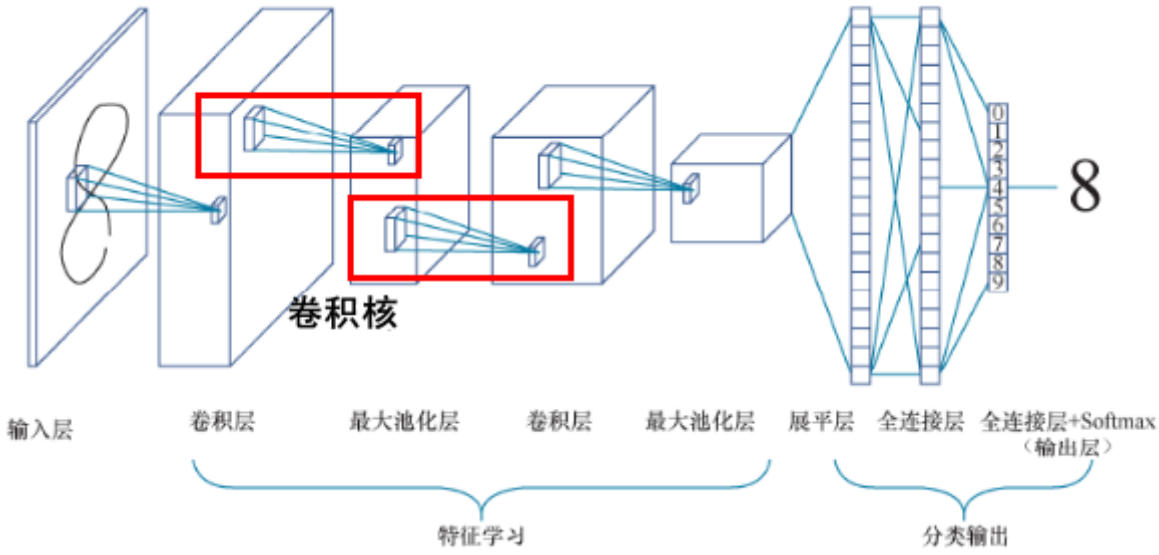
我们知道，在传统机器学习领域中，工具包 sklearn 是一枝独秀，但深度学习框架就不一样了，在这个领域呈现的是 TensorFlow 和 PyTorch 两强争霸的格局。这二者都是开源项目，一个来自 Google，一个来自 Facebook。PyTorch 因为编辑开发环境更具亲和力，支持快速和动态的训练，现在越来越受学术界和研究型开发者的欢迎，而 TensorFlow 则因为可以直接部署机器学习模型，能快速地开发和构建 AI 相关产品，它仍然在保持着工业界的霸主地位。

TensorFlow 的另一个优势是有简单的内置高级 API，这个 API 就是非常有名的 Keras，这也是初学者进入深度学习领域的最佳入门选择。Keras 把 TensorFlow 的底层深度学习功能进行了良好的封装，是最好用、最适合初学者上手的深度学习工具包了。所以，我们下面就选择 Keras 来搭建神经网络。

2. 建立模型

在搭建卷积神经网络之前，我们先来简单了解一下卷积网络是怎么一回事，这样我们在用代码构建时，就能清楚地知道每一步在干什么了。这里会出现一些新名词、新概念，不过你也不用害怕，因为我们并不是用编码来实现每一层，而是通过调用 API 来构建它们，你只要了解基本的结构框架，以及每一层的功能是什么，就足够了。

一个典型的卷积网络结构如下所示，它实现了一个图像分类功能：输入的是图像，输出的是图像的类别标签。



卷积网络的结构图

整体来看，卷积神经网络由输入层、一个或多个卷积层和输出层的全连接层组成。下面，我们按照图中从左往右的顺序，逐一来了解。

网络左边是数据输入部分，也就是输入层。这一层会对数据做初始处理，比如标准化、图片压缩、降维等，让最初的数据集变成形状为（ 样本，图像高度，图像宽度，颜色深度）的数据集。

接着，到了中间的卷积层。这一层主要负责抽取图片的特征，其中的卷积核（上图中红框部分）也叫滤波器，能够自动进行图像特征的提取。一般卷积层之后会接一个池化层，主要用来降低特征空间的维度，其中，池化层又包括最大池化和平均池化，它们的区别就在于输出时计算图片区域池化窗口内元素的最大值还是平均值。

通常，卷积 + 池化的架构会重复几次，形成深度卷积网络。在这个过程中，图片特征张

量的尺寸通常会逐渐减小，而深度将逐渐加深。就像在图中画的那样，特征图从一张扁扁的纸片形状变成了胖胖的矩形。

之后是一个展平层，主要负责将网络展平。展平之后通常会接一个普通的全连接层。而最右边的输出层也是全连接层，用 Softmax 进行激活分类输出层，所有神经网络都是用 Softmax 做多分类的激活函数。

卷积网络的核心特点就是“卷积 + 池化”的架构，而“卷积层”中的参数，其实是远少于全连接层的。这是因为卷积网络中各层的神经元之间，包括输入层的特征和卷积层之间，不是彼此全部连接的，而是以卷积的方式有选择性的局部连接。这种结构除了能大大减少参数的数量之外，还有有利于对图像特征的提取。

说完了卷积网络的结构和原理，现在我们用 Keras 来建立卷积神经网络模型。因为 Tensorflow 和 Keras 完全集成在 Kaggle 的 Notebook 中了，所以你不用 pip install 它们。我们直接调用其中的 API，就能够搭建起网络模型来。下面这段不到 20 行的代码，就为我们搭建起了一个能够为花朵图片分类的卷积神经网络：

复制代码

```
1 from tensorflow.keras import layers # 导入所有层 行1
```

```
from tensorflow.keras import models # 导入所有模型 行2
cnn = models.Sequential() # 贯序模型 行3
cnn.add(layers.Conv2D(32, (3, 3), activation='relu', # 输入卷积层 行4
                      input_shape=(150, 150, 3)))
cnn.add(layers.MaxPooling2D((2, 2))) # 最大池化层 行5
cnn.add(layers.Conv2D(64, (3, 3), activation='relu')) # 卷积层 行6
cnn.add(layers.MaxPooling2D((2, 2))) # 最大池化层 行7
cnn.add(layers.Conv2D(128, (3, 3), activation='relu')) # 卷积层 行8
cnn.add(layers.MaxPooling2D((2, 2))) # 最大池化层 行9
cnn.add(layers.Conv2D(128, (3, 3), activation='relu')) # 卷积层 行10
cnn.add(layers.MaxPooling2D((2, 2))) # 最大池化层 行11
cnn.add(layers.Flatten()) # 展平层 行12
cnn.add(layers.Dense(512, activation='relu')) # 全连接层 行13
cnn.add(layers.Dense(4, activation='softmax')) # 分类输出层 行14
cnn.compile(loss='categorical_crossentropy', # 损失函数 行15
            optimizer='RMSprop', # 优化器
            metrics=['acc']) # 评估指标
```

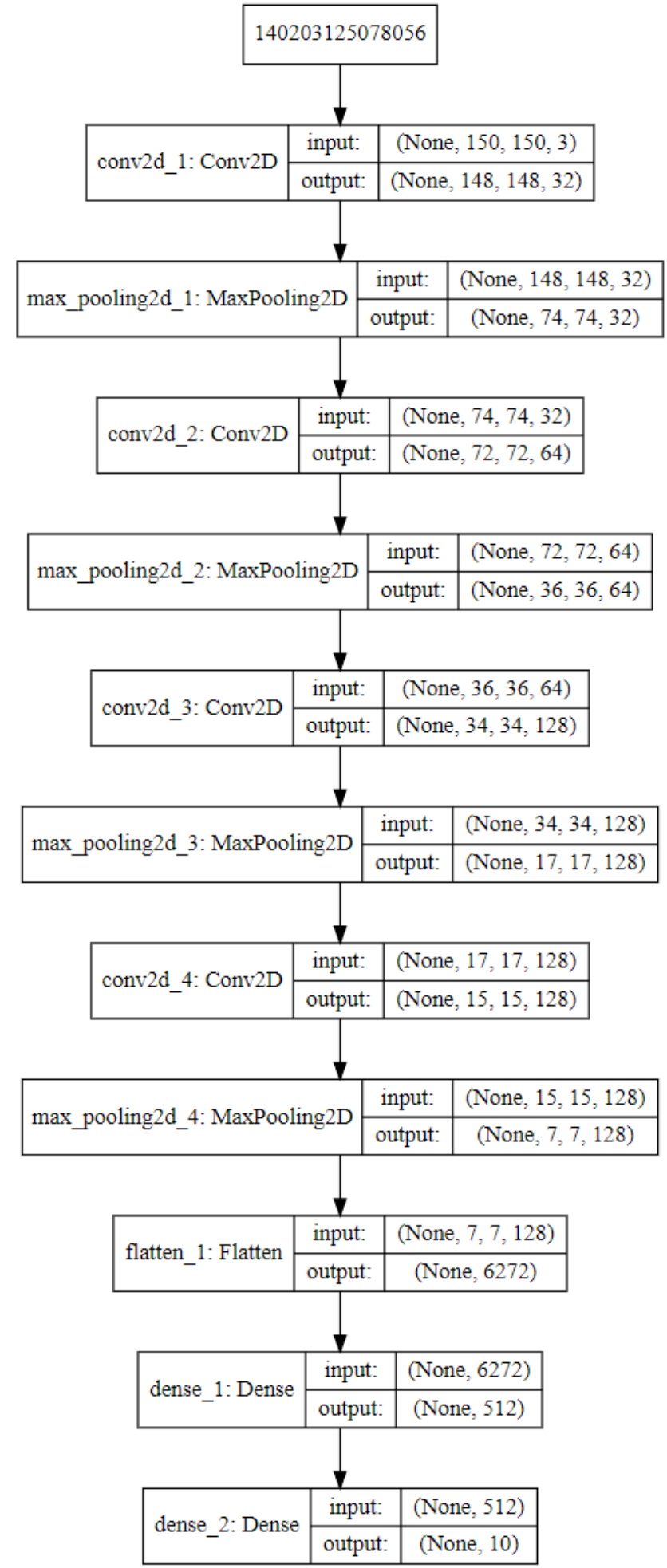
怎么样，这段代码的结构看起来也不复杂吧。其实神经网络中最主要的结构就是“层”，各种各样不同的层像拼积木一样组合起来，就形成了各种各样的神经网络。而对于我们的卷积神经网络 CNN 来说，其中最重要的就是 Conv2D 这个卷积层，它是我们这个神经网络的主要功能层，决定了我们所构建的神经网络是一个卷积神经网络。

现在我们一起来看看我们刚才搭建的这个 CNN 网络是个什么样子的结构，你可以用下面的方法来图片化显示这整个的 CNN 神经网络模型：

复制代码

```
from IPython.display import SVG # 实现神经网络结构的图形化显示
from tensorflow.keras.utils import model_to_dot # 导入model_to_dot工具
SVG(model_to_dot(cnn).create(prog='dot', format='svg')) # 绘图
```

输出如下：


140203125078056
conv2d_1: Conv2D
input: (None, 150, 150, 3)
output: (None, 148, 148, 32)
max_pooling2d_1: MaxPooling2D
input: (None, 148, 148, 32)
output: (None, 74, 74, 32)
conv2d_2: Conv2D
input: (None, 74, 74, 32)
output: (None, 72, 72, 64)
max_pooling2d_2: MaxPooling2D
input: (None, 72, 72, 64)
output: (None, 36, 36, 64)
conv2d_3: Conv2D
input: (None, 36, 36, 64)
output: (None, 34, 34, 128)
max_pooling2d_3: MaxPooling2D
input: (None, 34, 34, 128)
output: (None, 17, 17, 128)
conv2d_4: Conv2D
input: (None, 17, 17, 128)
output: (None, 15, 15, 128)
max_pooling2d_4: MaxPooling2D
input: (None, 15, 15, 128)
output: (None, 7, 7, 128)
flatten_1: Flatten
input: (None, 7, 7, 128)
output: (None, 6272)
dense_1: Dense
input: (None, 6272)
output: (None, 512)
dense_2: Dense
input: (None, 512)
output: (None, 10)

那么这个结构是怎么用刚才的 Keras API 搭建出来的呢？下面我就给你说一说，以后你自己搭建新的神经网络，也就是这样照猫画虎，搭积木似地往上构建就行了。

首先，我们通过 cnn = models.Sequential() 创建一个序贯模型（代码行 3）。序贯模型也是最简单的模型，就是像盖楼一样，一层一层往上堆叠着搭新的层。我们所有神经网络都用序贯模型，还有一种模型是函数式模型，是比较高级地技巧，不过比较少见，你如果有兴趣可以查阅资料去了解。

然后，我们通过 cnn.add(layers.Conv2D(32, (3, 3), activation= ‘relu’, input_shape= (150, 150, 3))) 语句（代码行 4）在模型中加入了神经网络的输入层。这里面的激活函数 relu 我们以后再解释，现在你只要知道 relu 是一个常用的激活函数就可以了。这里，你需要注意的是，输入层需要通过 input_shape=(150, 150, 3) 指定输入的特征数据集的形状。这个刚才我们已经说过了，如果形状不对，等会儿拟合时就会报错。

接着，从代码行 5 开始直到行 13，我们为神经网络添加了各种各样的中间层（也叫隐层），这些层如何添加、配置，我们有足够的自由去尝试。少可以两三层，多可以几万层。

其中，cnn.add(layers.Conv2D(64, (3, 3), activation= ‘relu’))（代码行 6）这个语句是用来添加输入层之后的中间卷积层的，这里面的 64 是输出空间的维度，也就是卷积过程中输出滤波器的数量，而 (3, 3) 则指定 2D 卷积窗口的高度和宽度。
cnn.add(layers.MaxPooling2D((2, 2)))（代码行 7）这个语句是用来添加池化层，(2, 2) 也是指定 2D 卷积窗口的高度和宽度。

以此类推，卷积层 + 池化层的组合会持续出现，然后再输出层之前需要有展品层（代码行 12）和全连接层（代码行 13）。

我们接着看代码行 14，cnn.add(layers.Dense(10, activation= ‘softmax’)) 这一行叫做输出层，其中 activation='softmax’ 这个参数，就用于多分类输出。这个暂时你就记住，先不用去管细节，讲分类问题的时候我们再详谈。

那么你可能会问，怎么看一个神经网络是普通神经网络 DNN，还是 CNN 或者 RNN 呢？这其中的关键就是看输入层和中间层主要是什么类型。DNN 的输入层和中间层主要是

Dense 层，CNN 的输入层和中间层主要是 Conv1D、Conv2D 或者 Conv3D，RNN 的输入层和中间层主要是 SimpleRNN 或者 GRU 或者 LSTM 层。

怎么样，讲到这里，你明白怎么去搭建神经网络了吗？我知道，如果你是第一次接触深度学习的话，这里的新概念的确有点多，不过，这里其实也并没有什么很难懂的东西，你需要把选择算法建立模型这个环节，反复阅读，咀嚼几遍，就可以掌握搭建神经网络模型的方法。

模型的训练和拟合

模型构建好之后，我们用非常熟悉的 fit 语句来进行训练：

复制代码

```
# 训练网络并把训练过程信息存入history对象
history = cnn.fit(X_train,y_train, #训练数据
                  epochs=10, #训练轮次（梯度下降）
                  validation_split=0.2) #训练的同时进行验证
```

在训练过程中，我们还指定了 validation_split，它可以在训练的同时，自动把训练集部分拆出来，进行验证，在每一个训练轮次中，求出该轮次在训练集和验证集上面的损失和预测准确率。

训练输出如下：

复制代码

```
Train on 2089 samples, validate on 523 samples
Epoch 1/5
2089/2089 [==============================] - 86s 41ms/step - loss: 1.3523 - ac
Epoch 2/5
2089/2089 [==============================] - 85s 41ms/step - loss: 1.0167 - ac
Epoch 3/5
2089/2089 [==============================] - 85s 41ms/step - loss: 0.8912 - ac
Epoch 4/5
2089/2089 [==============================] - 84s 40ms/step - loss: 0.8295 - ac
Epoch 5/5
2089/2089 [==============================] - 85s 41ms/step - loss: 0.7228 - ac
... ...
```

这个输出的信息包括了训练的轮次（梯度下降的次数）、每轮训练的时长、每轮训练过程中的平均损失，以及分类的准确度。这里的每一个轮次，其实就是神经网络对其中的每一个神经元自动调参、通过梯度下降进行最优化的过程。

模型性能的评估

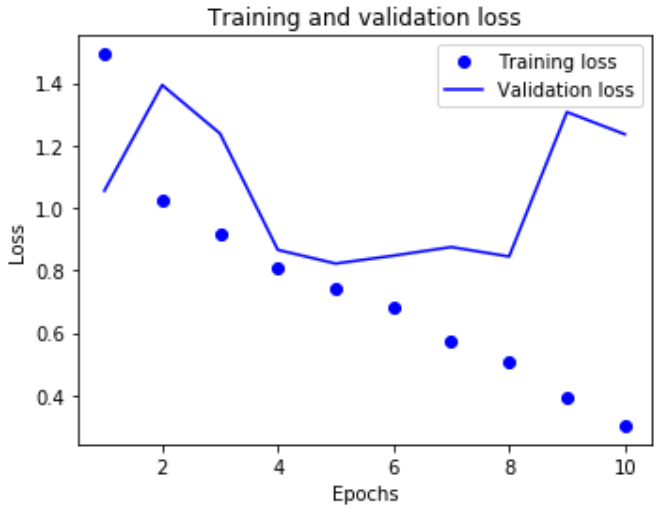
刚才的训练过程已经包含了验证的环节。不过，为了更好地体现训练过程中的损失变化情况，我们这里把每轮的损失和准确率做一个可视化，绘制出损失曲线，来展示模型在训练集上评估分数和损失的变化过程。

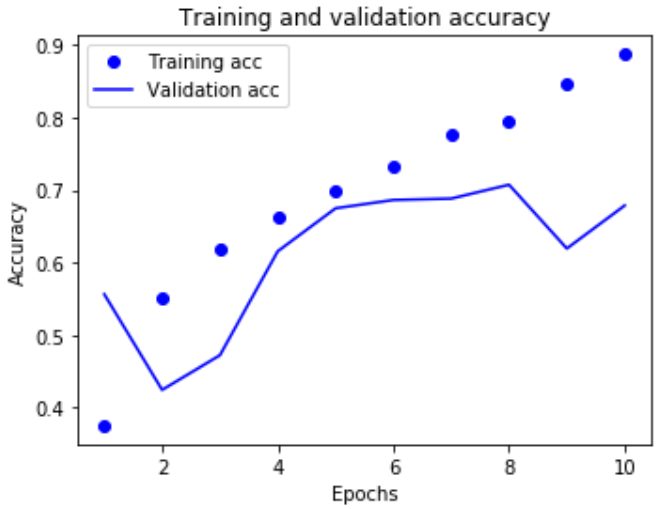
这种损失曲线其实你并不陌生，在第 6 讲中，我们在讲 K-Means 算法中的手肘图时候就绘制过类似的曲线：

复制代码

```
def show_history(history): # 显示训练过程中的学习曲线
    loss = history.history['loss'] #训练损失
    val_loss = history.history['val_loss'] #验证损失
    epochs = range(1, len(loss) + 1) #训练轮次
    plt.figure(figsize=(12,4)) # 图片大小
    plt.subplot(1, 2, 1) #子图1
    plt.plot(epochs, loss, 'bo', label='Training loss') #训练损失
    plt.plot(epochs, val_loss, 'b', label='Validation loss') #验证损失
    plt.title('Training and validation loss') #图题
    plt.xlabel('Epochs') #X轴文字
    plt.ylabel('Loss') #Y轴文字
    plt.legend() #图例
    acc = history.history['acc'] #训练准确率
    val_acc = history.history['val_acc'] #验证准确率
    plt.subplot(1, 2, 2) #子图2
    plt.plot(epochs, acc, 'bo', label='Training acc') #训练准确率
    plt.plot(epochs, val_acc, 'b', label='Validation acc') #验证准确率
    plt.title('Training and validation accuracy') #图题
    plt.xlabel('Epochs') #X轴文字
    plt.ylabel('Accuracy') #Y轴文字
    plt.legend() #图例
    plt.show() #绘图
show_history(history) # 调用这个函数
```

损失曲线输出如下：





可以看到，训练集的损失呈现下降趋势，但是测试集上的损失则呈现跳跃，这说明这个神经网络性能不是很稳定，似乎有过拟合的现象。我们将在后面的讲解中探讨如何优化神经网络，让神经网络的损失值更低，准确率更高。

最后，我们用这个训练好了的模型，在测试集上进行分类结果的评分。

复制代码

```
result = cnn.evaluate(X_test, y_test) #评估测试集上的准确率
print('CNN的测试准确率为',"{0:.2f}%".format(result[1]))
```

输入如下：

复制代码

```
653/653 [==============================] - 10s 15ms/step
CNN的测试准确率为 0.69%
```

输出显示，在 653 张测试集的图片中测试，模型的分类准确率达到了 0.69 以上。

最后呢，我们可以应用模型的 predict 属性把 X 特征集传入，进行花朵图片的分类。

复制代码

```
prediction = cnn.predict(X_test) #预测测试集的图片分类
```

下面的代码我们输出第一个图片（Python 是从 0 开始索引的，所以下标 0 就是第一张图片）的分类预测值：

```
prediction[0] #第一张图片的分类
```

复制代码

输出如下：

```
array([0.0030566 , 0.13018326, 0.00846946, 0.8582906 ], dtype=float32)
```

复制代码

这里请你注意，此时输出的是分类概率。上面的输出结果表示，第一类花 Daisy（雏菊）的概率为 0.03，第二类花 Rose（玫瑰）的概率为 0.13，第三类花 Sunflower（向日葵）的概率为 0.008，第四类花 Tulip（郁金香）的概率为 0.858。

下面的代码中，我们选出最大的那个概率，并把它当作 CNN 的分类结果：

```
print('第一张测试图片的分类结果为:', np.argmax(prediction[0]))
```

复制代码

输出如下：

```
第一张测试图片的分类结果为: 3
```

复制代码

结果显示第一个图片被 CNN 网络模型分类为第 4 种花（索引从 0 开始，所以类别 3 就是第 4 种花），也就是 Tulip（郁金香）。

好啦，这个神经网络完成了很好的分类功能，准确率达到 69%！这比我可强多了，反正我是区分不出来这些花的种类的。那么，神经网络的超参数我们还不知道是怎么一会事，也不知道如何继续优化。不必担心，今天只是讲了个基础。关于性能的超参数的具体用户和性能调优，我们会找一个时间单独讲解的。

总结一下

怎么样，构建强大的深度学习神经网络，并没有你想象得那么难吧？

在这一讲中，我带你去了一个非常好用的网站 Kaggle，根据这个网站中的花朵图片数据集创建了 Jupyter Notebook，下面几节课中，我们都需要利用 Kaggle 网站中的 GPU 等资源来加速网络模型的拟合速度。

我们这节课的重点在于了解卷积神经网络的结构以及它的搭建方法，它由输入层、一个或多个卷积层和全连接输出层组成。要构建出能够实现图像分类的 CNN，我们通常需要把“卷积 + 池化”的组合重复搭建几次，形成深度卷积网络。在这个过程中，图片特征的尺寸通常会逐渐减小，而深度将逐渐加深。然后我们把它展平，之后通常接一个普通的全连接层，再接一个分类输出层。

这个过程特别像搭积木，全都是通过 Keras 的 add API，添加 layers 中所提供的各种类型的层来完成的。如果你希望搭建更深层的神经网络，只要加入更多的卷积层，或者加入更多的全连接层就可以，这个过程并无一定之规，你可以随意尝试。

当然如果你的层与层之间组合有误，或者输入层所指定接收的张量格式和 X 特征集的形状不匹配，编译的时候就会报错。这也没什么大不了的，就像我们拿起了两个接口不一样乐高玩具，当然无法拼合在一起。你只要根据错误信息，按图索骥，找到合适的组合方案就可以了。

思考题

这节课到这里就讲完了，我来给你留一道练习题：

今天我们搭建出了一种形状的神经网络，我刚才也说了，在搭建网络结构的时候，我们就像个建筑师一样，可以自行组合层的数量和各层的样式。那么，现在请你改变一下网络的结构，自己搭建新的网络，比较一下性能。

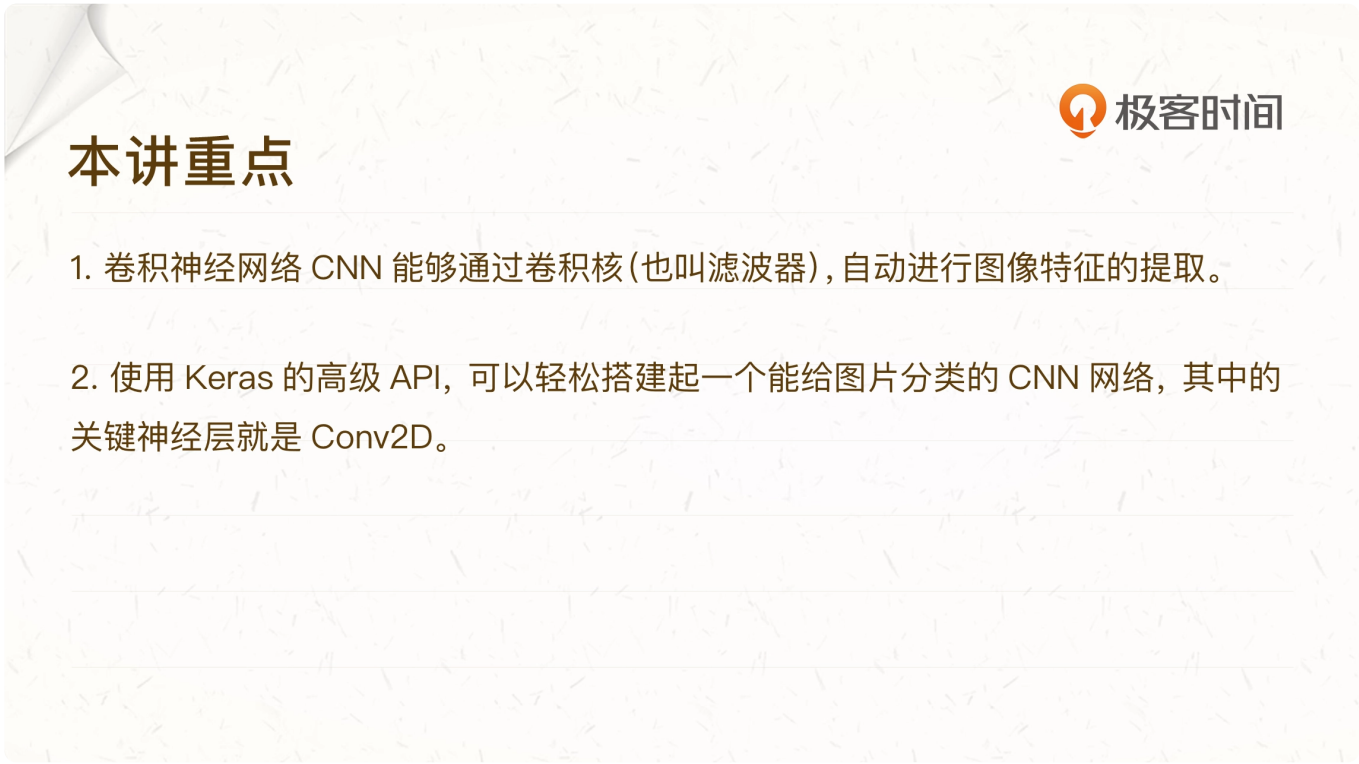
提示：你可以在我们今天学的 CNN 网络的基础上：

1. 增加更多的卷积层、池化层组合；
2. 增加或减少中间卷积层滤波器的个数（如 64，128，256，512...）；

3. 调整中间层中卷积和池化窗口的高度和宽度（如 (5, 5)、(7, 7)...）

```
cnn.add(layers.Conv2D(256, (5, 5), activation= ‘relu’ )) # 卷积
cnn.add(layers.MaxPooling2D((3, 3))) # 最大池化
```

欢迎你积极留言提交属于你自己的神经网络模型的评估分数，看看谁的模型更棒！我们下一讲见！



分享给需要的人，Ta订阅后你可得 20 元现金奖励

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 10 | 模型优化（下）：交叉验证，同时寻找最优的参数

下一篇 12 | 深度学习（中）：如何用RNN预测激活率走势？

精选留言 (3)

写留言

在路上

2021-09-22

佳哥好，今天的课程帮助我了解了深度学习的整个流程，虽然还不知道卷积和池化的参数为什么是这样设置的，但是图示的过程很容易理解。体验了下kaggle，新建notebook之后，一直处于editor loading状态。

展开

作者回复: Kaggle今天出了点小问题，现在好啦。

davidhuxiangwei

2021-09-22

老师，我用kaggle 的 notebook，打开后一直出现 editor loading ，所有操作按钮都不能用，是什么原因呢？

展开

作者回复: Kaggle今天出了点小问题，现在好啦。

aaaaa

2021-09-22

好多概念能够更加清晰

展开